CS 194: Distributed Systems
Distributed File Systems

Scott Shenker and Ion Stoica
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720-1776

Outline

- Network File System (NFS)
- CODA

Main Goal

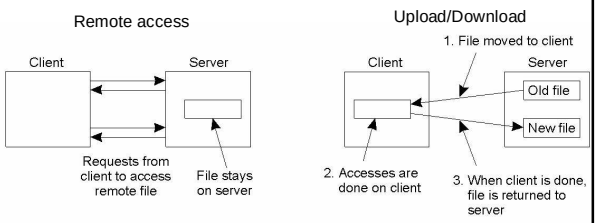
- Provide a client **transparent** access to a file system stored at a **remote** server
- Why would you want to store files remotely?

NFS

- A specification for a distributed file system (by Sun, 1984)
- Implemented on various OS's
- De facto standard in the UNIX community
- Latest version is 4 (2000)
- Client-server file system

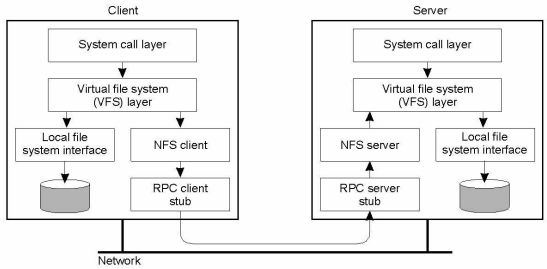
Access Model

- Two access models :
  - Remote access
  - Upload/Download



NFS Architecture

- Virtual File System (VFS) provide a uniform access to local and remote files



File System Model

- Similar to UNIX: files are treated as uninterpreted sequences of bytes
- Each files has a name, but usually referred by a **file handle**
  - Client use a name service to get file handle
- Files organized into a naming graphs
  - Nodes → directories or files
- First three versions were **stateless**; version 4 is **stateful**

7

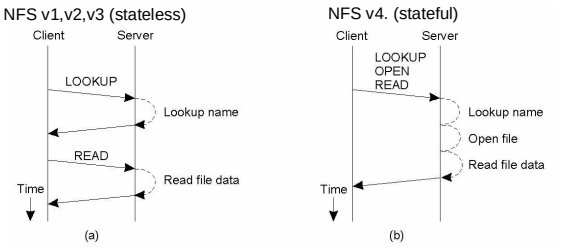
Stateful vs. Stateless

- Stateless model: each call contains complete information to execute operation
- Stateful model: server maintain context (info) shared by consecutive operations
- Discussion: compare stateless and stateful design

8

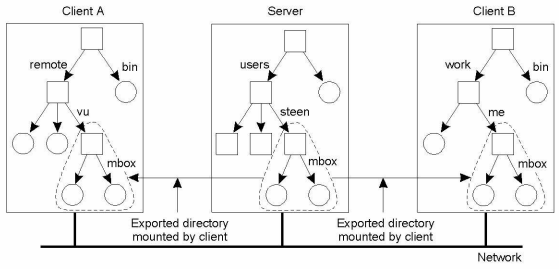
Communication

- RPC based
- One operation per RPC (NFS v. 1,2,3)
- Multiple operations per RPC (NFS v. 4)

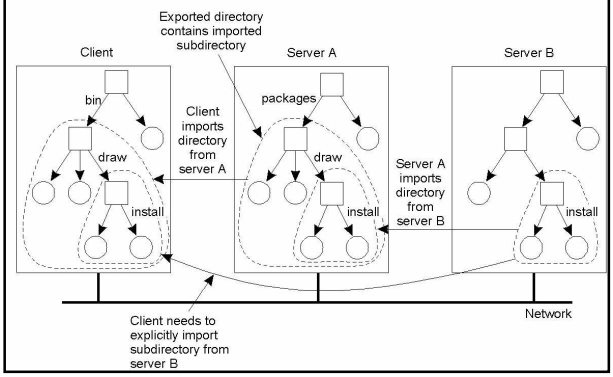


Naming

- Allow a client to mount a remote file system into its own local file system
- Pathnames are **not** globally unique; what's the implication?



Example: Mounting Nested Directories



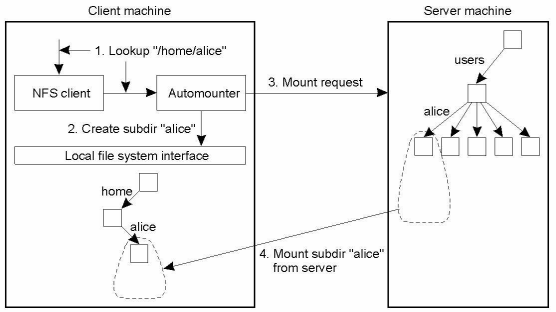
File Handles

- File handle: created by server hosting the file
- Unique with respect to all file systems exported by servers
- Persistent: doesn't change during file's lifetime
- Length: 32b in v2, 64b in v3, and 64b in v4

12

Automounting

- Mount file system **transparently** when client accesses it



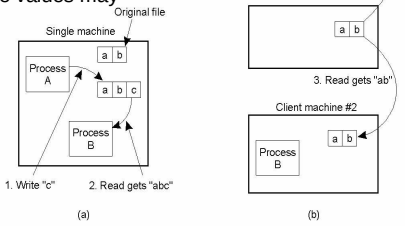
Mandatory File Attributes

Attribute	Description
TYPE	The type of the file (regular, directory, symbolic link)
SIZE	The length of the file in bytes
CHANGE	Indicator for a client to see if and/or when the file has changed
FSID	Server-unique identifier of the file's file system

Semantics of File Sharing

a) On a single processor, when a *read* follows a *write*, the value returned by the *read* is the value just written

b) In a distributed system with caching, obsolete values may be returned.



Semantics of File Sharing

Method	Comment
UNIX semantics	Every operation on a file is instantly visible to all processes
Session semantics	No changes are visible to other processes until the file is closed
Immutable files	No updates are possible; simplifies sharing and replication
Transaction	All changes occur atomically

- NFS implements session semantics

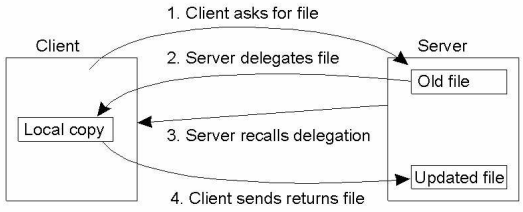
File Locking in NFS

- NFS v1-v3: use a separate (stateful) lock manager
- NFS v4: integrated in the file system

Operation	Description
Lock	Creates a lock for a range of bytes
Lockt	Test whether a conflicting lock has been granted
Locku	Remove a lock from a range of bytes
Renew	Renew the leas on a specified lock

Client Caching

- NFS v1-v3: mainly left outside the protocol (see book)
- NFS v4: use file delegation



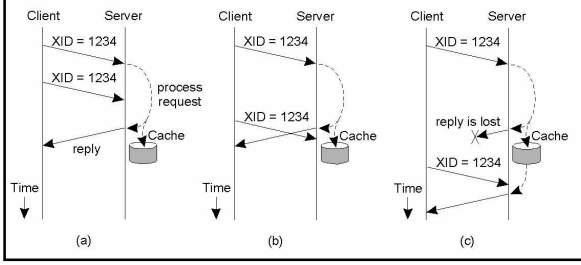
Fault Tolerance

- Need to maintain state consistent in v4, e.g.,
  - Locking
  - Delegation
- Challenge: eliminate duplicate operations in case of failure
- Solution: use transaction identifiers (XID)

19

Handling Retransmissions

a) Request still in progress
b) Reply has just been returned
c) Reply has been some time ago, but was lost

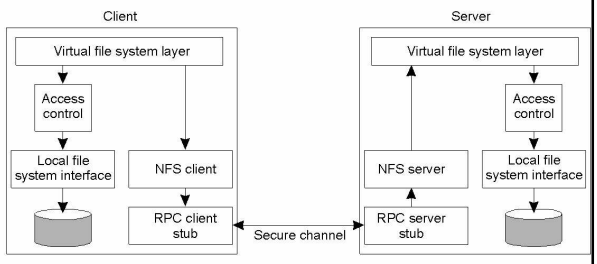


Security

- Secure RPC: three methods
  - System authentication: trust the user has passed a proper login procedure
  - Diffie-Hellman key exchange (but not very secure—uses only 192b Keys)
  - Kerberos
- File access control
  - Use access control list (ACL)

21

Security Architecture



22

Outline

- Network File System (NFS)
- CODA

23

The Coda File System

- Developed at CMU
- Based on Andrew File System (AFS), another distributed system developed at CMU
  - Community wide system

24

AFS Goals

- **Scalability:** system should grow without major problems
- **Fault-Tolerance:** system should remain usable in the presence of server failures, communication failures and voluntary disconnections
- **Unix Emulation**
- **Design philosophy: Scalability and Accessibility more important than consistency**

25

Coda Goals

- AFS goals, plus
- **Disconnected mode** for **portable computers**

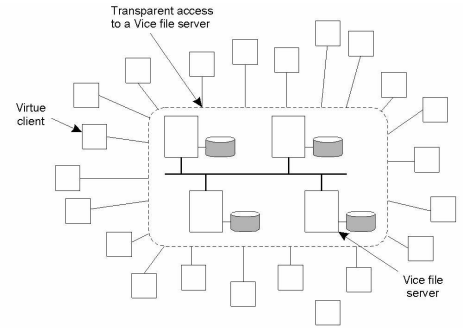
26

System Model

- Client workstations are personal computers owned by their users
  - **Fully autonomous**
  - **Cannot be trusted**
- Coda allows laptops that operate in **disconnected mode**

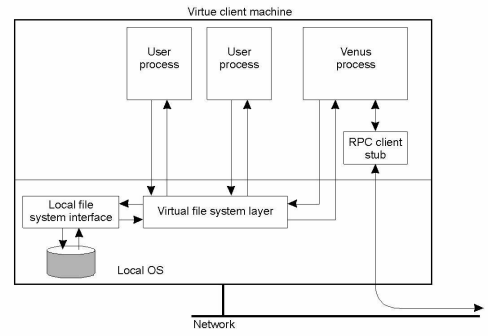
27

Overall Organization of AFS & Coda



28

Internal Organization of Virtue



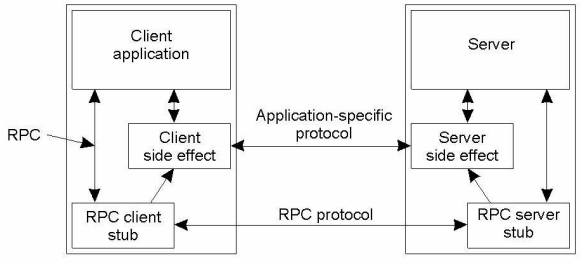
29

Communication

- Based on RPC2: provides reliable transmission on top of UDP
- RPC2 supports side-effects, i.e., user defined protocols
- RPC2 provides support for multicast
  - Transparent for the client

30

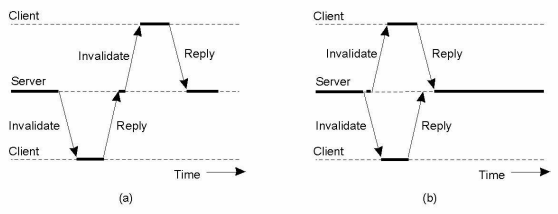
Side Effects in Coda's RPC



31

Support for Multicast in RCP2

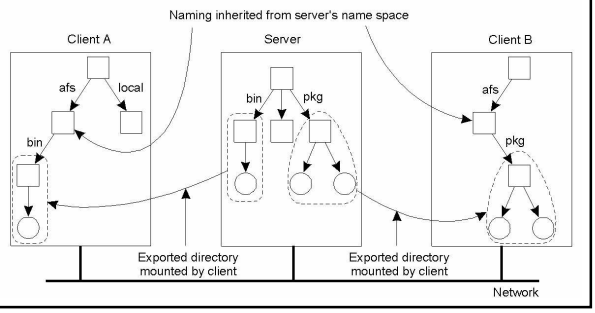
- Example: send invalidation
  a) One at a time
  b) Multicast



32

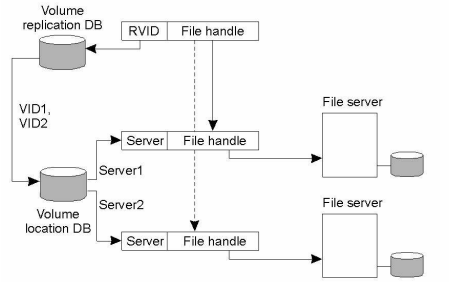
Naming

- Single shared naming space (vs. client-based in NFS)



File Identifiers

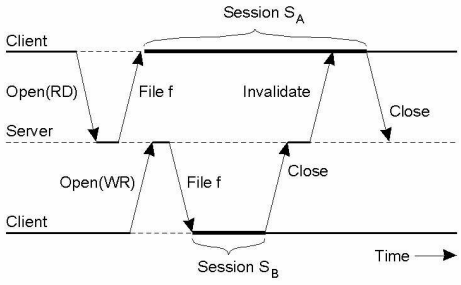
- Globally unique (vs server unique in NFS)
- Coda distinguishes between physical and logical volumes
- Logical volume (identified by RVID) a possible replicated physical volume (identified VID)



34

Sharing Files in Coda

- Transaction semantics
  - Session is treated like a transaction



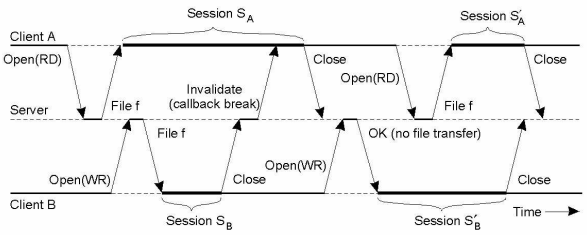
35

Caching and Replication

- Caching:
  - Achieve scalability
  - Increases fault tolerance
- Challenge: how to maintain data consistency?
- Solution: use callbacks to notify clients when a file changes
  - If a client modifies a copy, server sends a **callback break** to all clients maintaining copies of same file

36

Example: Client Caching

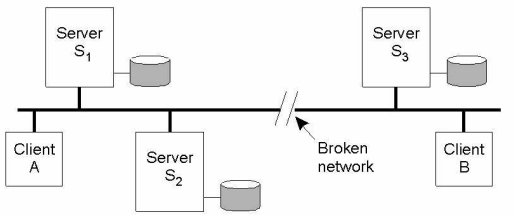


Server Replication

- Unit of replication: volume
- Volume Storage Group (VSG): set of servers that have a copy of a volume
- Accessible Volume Storage Group (AVSG): set of servers in VSG that the client can contact
- Use vector versioning
  - One entry for each server in VSG
  - When file updated, corresponding version in AVSG is updated

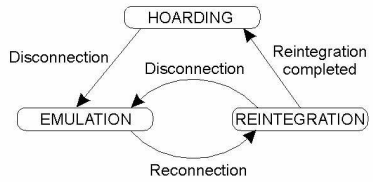
Example: Handling Network Partition

- Versioning vector when partition happens: [1,1,1]
- Client A updates file → versioning vector in its partition: [2,2,1]
- Client B updates file → versioning vector in its partition: [1,1,2]
- Partition repaired → compare versioning vectors: **conflict**!



Disconnected Operation

- HOARDING: File cache in advance with all files that will be accessed when disconnected
  - Best effort
- EMULATION: when disconnected, behavior of server emulated at client
- REINTEGRATION: transfer updates to server; resolves conflicts

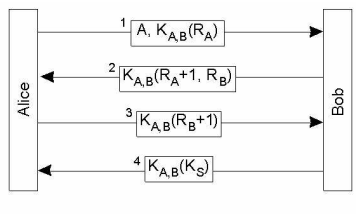


Security

- Set-up a secure channel between client and server
  - Use secure RPC
- System-level authentication

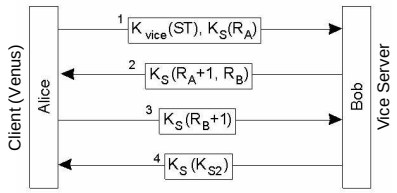
Mutual Authentication in RPC2

- Based on Needham-Schroeder protocol



Establishing a Secure Channel

- Upon authentication AS (authentication server) returns:
  - Clear token: CT = [Alice, TID, K_S, T_{start}, T_{end}]
  - Secret token: ST = $K_{vice}([CT]^*_{Kvice})$
  - K_S: secret key obtained by client during login procedure
  - K_{vice}: secret key shared by vice servers
- Token is similar to the ticket in Kerberos



NFS vs Coda

Issue	NFS	Coda
Design goals	Access transparency	High availability
Access model	Remote	Up/Download
Communication	RPC	RPC
Server groups	No	Yes
Mount granularity	Directory	File system
Name space	Per client	Global
Sharing sem.	Session	Transactional
Cache consist.	write-back	write-back
Fault tolerance	Reliable comm.	Replication and caching
Recovery	Client-based	Reintegration
Secure channels	Existing mechanisms	Needham-Schroeder